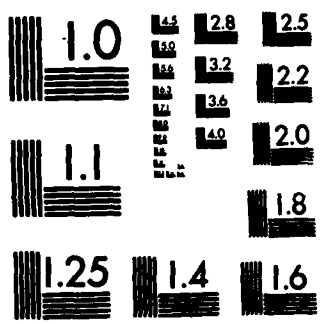
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

An improved algorithm for the rectangle enclosure problem+

D. T. Lee and F. P. Preparata

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A124 461
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) STEP INTO COMPUTATIONAL GEOMETRY NOTEBOOK III	5. TYPE OF REPORT & PERIOD COVERED Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) F. P. Preparata	8. CONTRACT OR GRANT NUMBER(s) MCS-78-13642;MCS-79-16847; N00014-79-C-0424;DAAG-29-78-C-0016; CDCI-04-AB
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory, 1101 W. Springfield, University of Illinois at Urbana-Champaign Urbana, Illinois 61801	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS National Science Foundation; Joint Services Electronics Program;	12. REPORT DATE May 1981
	13. NUMBER OF PAGES 29
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computational geometry; analysis of algorithms; computational complexity; monotone polygon; rectangle enclosure; vector dominance; shortest path; constrained shortest path

theta

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this notebook we present a collection of three new results in planar computational geometry. The first problem is to test a given n-vertex simple polygon for monotonicity; this problem can be optimally solved in time $\theta(n)$. The second result is an improved algorithm for the rectangle enclosure problem; this algorithm improves over an existing one by using optimal space $\theta(n)$. Finally, the third result is the construction, in time O(nlogn), of the shortest path between two points in the interior of an n-vertex polygon P, when the path is constrained to lie within P.

DD FORM 1 JAN 73 1473

STEP INTO COMPUTATIONAL GEOMETRY

NOTEBOOK III

by

Franco P. Preparata

This work was supported in part by the National Science Foundation and in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) and in part by the Control Data Corporation.

STEP INTO COMPUTATIONAL GEOMETRY

NOTEBOOK III

Abstract

In this notebook we present a collection of three new results in planar computational geometry. The first problem is to test a given n-vertex simple polygon for monotonicity; this problem can be optimally solved in time $\theta(n)$. The second result is an improved algorithm for the rectangle enclosure problem; this algorithm improves over an existing one by using optimal space $\theta(n)$. Finally, the third result is the construction, in time $O(n\log n)$, of the shortest path between two points in the interior of an n-vertex polygon P, when the path is constrained to lie within P.

STEPS INTO COMPUTATIONAL GEOMETRY

NOTEBOOK III

F. P. Preparata, Editor

A long while after Notebook II of this collection, which appeared in September 1977, this notebook contains a few new results in computational geometry, whose manuscript sizes do not reach the usual standard of technical reports but whose content may be of interest to researchers in the field. Again, one of the main reasons of this collection is ease of access.

The first problem considered is the test of whether a given n-vertex simple polygon is monotone. Since certain computational problems involving polygons are easier for monotone than for arbitrary simple polygons, the question has not only theoretical but also practical interest. We present an $\theta(n)$ time solution of the problem of deciding whether a given simple polygon P is monotone and, if so, of exhibiting a line ℓ with respect to which P is monotone. As a consequence, a monotone polygon can be triangulated also in time $\theta(n)$.

Next we have studied a problem which has received some attention recently in the context of the geometry of rectangles: the rectangle enclosure problem. Given a set of n rectangles in the plane, with sides parallel to the coordinate axes, we must find all q pairs of rectangles such that one rectangle of the pair encloses the other. The algorithm presented is an alternative to and an improvement of the one by Vaishnavi and Wood. While both techniques have worst-case running time $O(n\log^2 n+q)$, the described algorithm uses optimal storage $\theta(n)$ rather than $O(n\log^2 n)$ as the Vaishnavi-Wood's technique, and works entirely in-place using very conventional data structures.

The third problem reported in this notebook is the construction of the Euclidean shortest path within a simple polygon P. Given source s and destination t as two points in the interior of P, Shamos had originally solved this problem (which he called "internal distance") by first constructing the viewability graph of its vertices and subsequently by applying a standard shortest path algorithm to the viewability graph, where each edge is weighted with its length. In actuality only relevant portions of the viewability graph need be constructed. Here we present an algorithm based on the observation that if we triangulate P, the shortest path is topologically a path on the dual of the triangulated P. The described algorithm runs in time O(nlogn) for an n-vertex P.

TESTING A SIMPLE POLYGON FOR MONOTONICITY*

Franco P. Preparata and Kenneth J. Supowit

1. Introduction

Let P be a simple polygon in the plane having vertices $p_0, p_1, \ldots, p_{n-1}$ counterclockwise on its boundary. The sides of P, called arcs, are denoted as $e_j = (p_j, p_{j+1})$ and are directed from p_j to p_{j+1} (indices are taken modulo n throughout). A chain $C_{ij} = (e_i, e_{i+1}, \ldots, e_{j-1})$ is a sequence of arcs on the boundary of P. C_{ij} is monotone with respect to a (straight) line ℓ if the projections of the vertices $p_i, p_{i+1}, \ldots, p_j$ on ℓ are ordered as the vertices in C_{ij}. P is monotone if there exists a line ℓ such that the boundary of P can be partitioned into two chains C_{ij} and C_{ji} that are monotone with respect to ℓ (if a direction is chosen on ℓ then one chain is monotone non-decreasing, the other is monotone non-increasing).

Note that the class of monotone polygons properly contain the class of convex polygons, and are properly contained in the class of simple polygons. It appears that certain computational problems involving polygons are easier for monotone than for arbitrary simple polygons. For example, the fastest algorithm known to triangulate an arbitrary simple polygon

*This work was supported in part by National Science Foundation Grant MCS-78-13642, by the Joint Services Electronics Program Contract N00014-79-C-0424, and by Control Data Corporation Contract CDCI-04-AB.

requires $\Theta(n\log n)$ time [1]. However, given a line ℓ and a polygone P monotone with respect to ℓ, P can be triangulated in $\theta(n)$ time [1].

We consider the following problem: given a simple polygon P, decide whether P is monotone and, if so, exhibit a line ℓ with respect to which P is monotone. We present a $\theta(n)$ time solution to this problem; hence, by the above remarks, there is a $\theta(n)$ time algorithm that, given a simple polygon, triangulates it in $\theta(n)$ time if it is monotone.

2. The algorithm

Given the polygon P, as defined in the preceding section, let θ_i be the counterclockwise polar angle at arc e_i $(i = 0,\ldots,n-1)$ with respect to a chosen direction (for example, the direction of e_0). Define α_i as the counterclockwise wedge from θ_{i-1} to θ_i if the external angle at vertex p_i is $\geq 180°$; as the clockwise wedge from θ_{i-1} to θ_i, otherwise. Note that, by the simplicity of P, the angle of wedge α_i $(i = 0,\ldots,n-1)$ has size $< 180°$.

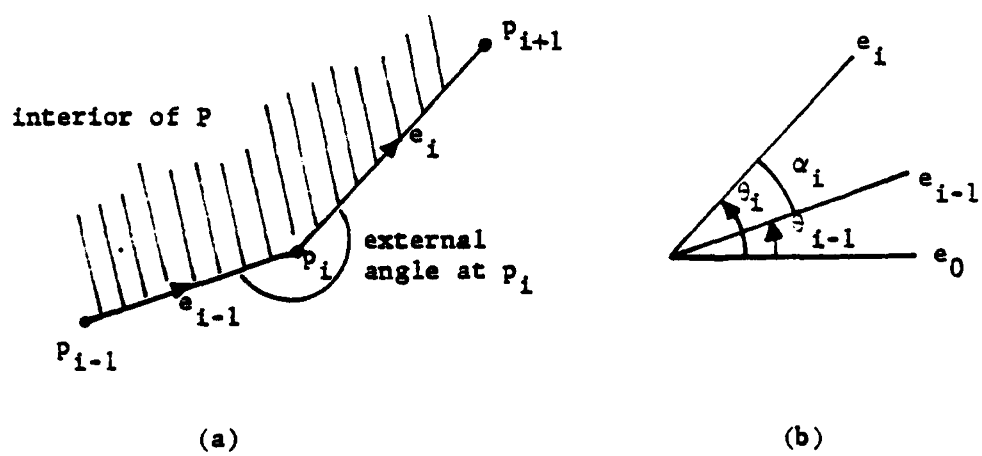

Figure 1. Illustration of the correspondence between wedge α_i and the external angle at p_i.

Given a chain $C = (e_j, e_{j+1}, \ldots, e_{j+k})$, define $\alpha(C) \triangleq \bigcup_{i=j+1}^{j+k} \alpha_i$, i.e., $\alpha(C)$ is the union of the wedges $\alpha_j, \alpha_{j+1}, \ldots, \alpha_{j+k-1}$. Obviously, $\alpha(C)$ is a wedge. We now prove:

<u>Lemma 1</u>. $C = (e_1, \ldots, e_k)$ is monotone with respect to ℓ if and only if the normal to ℓ has a polar angle $\vartheta \notin \alpha(C)$.

<u>Proof</u>: Given that C is monotone with respect to ℓ, suppose that $\vartheta \in \alpha(C)$ (figure 2b). This implies that there is at least one wedge α_i such that $\vartheta \in \alpha_i$, for some $i \in \{1,2,\ldots,k-1\}$. If we now consider (figure 2a) the

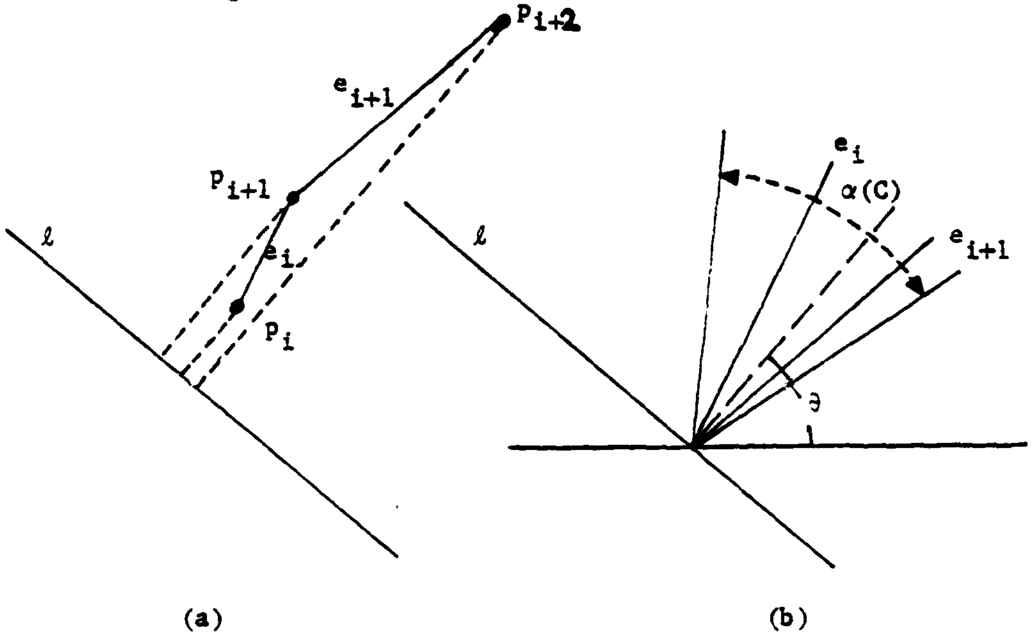

Figure 2. Illustration for the proof of Lemma 1.

projections of vertices p_i, p_{i+1}, and p_{i+2} on ℓ, we have that the projections of p_i and p_{i+2} are on the same side of that of p_{i+1}, i.e., C is not monotone with respect to ℓ, a contradiction. Thus $\vartheta \notin \alpha(C)$.

Conversely, suppose that C is not monotone with respect to ℓ. Then there is a vertex p_i of C for which the preceding arguments can be reversed.

$\square$

Consider now a monotone polygon P. Monotonicity means that there are two vertices p_i and p_j of P and a line ℓ, such that chains C_{ij} and C_{ji} are monotone with respect to ℓ (figure 3a). In the polar diagram (figure 3b), we construct the wedges $\alpha(C_{ij})$ and $\alpha(C_{ji})$. These two wedges are possibly separated by two wedges γ_j and γ_i (see figure 3b). Note that $\theta_{i-1} \in \alpha(C_{ji})$ and $\theta_i \in \alpha(C_{ij})$; also $\gamma_i \subseteq \alpha_i$, whence size $(\gamma_i) \le$ size $(\alpha_i) <$ 180°. Similarly, size $(\gamma_j) <$ 180°. Moreover, by Lemma 1, the line ℓ' passing through the origin of the polar diagram and perpendicular to ℓ intersects neither $\alpha(C_{ij})$ nor $\alpha(C_{ji})$; thus $\alpha(C_{ij})$ and $\alpha(C_{ji})$ lie on opposite sides of ℓ'.

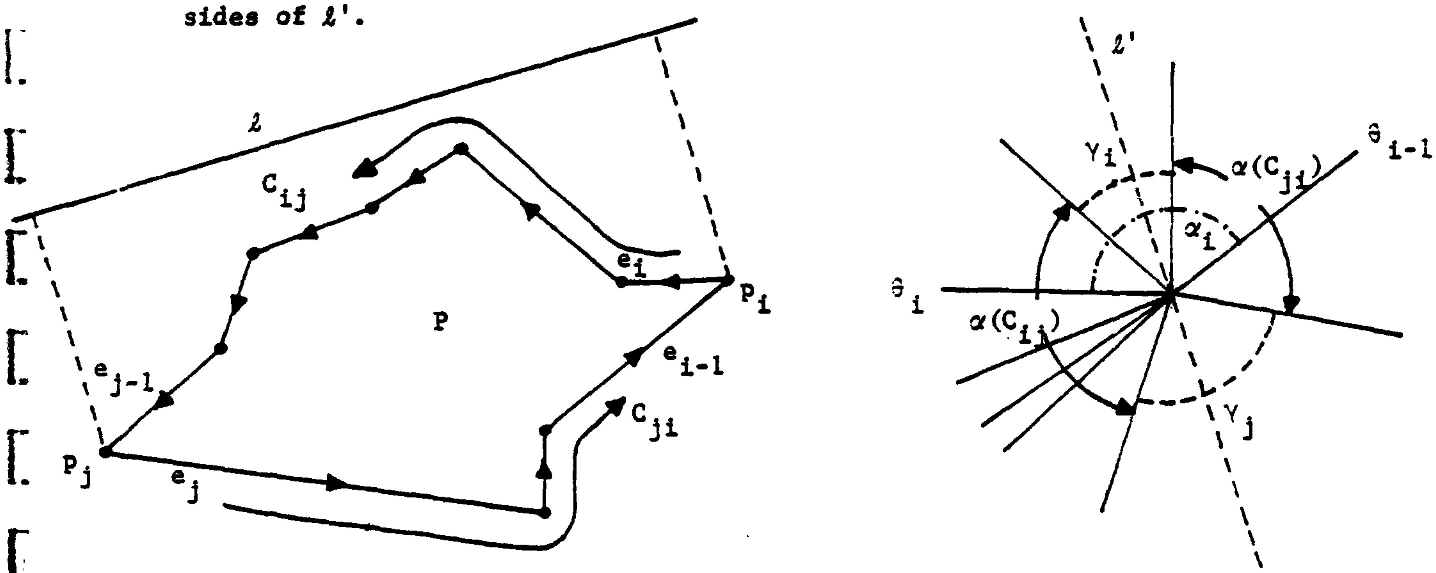

Figure 3. Illustration of the correspondence between a simple polygon P and polar diagram of its arcs.

The polar rays corresponding to the n arcs of P partition the polar range $[0,2\pi)$ into n consecutive wedges (some of which could be of size 0). Let · be one · these wedes; the <u>multiplicity</u> $\mu(\alpha)$ of α is defined as $\mu(\alpha)$ ^ ; $\{\alpha_i: \alpha \subseteq \alpha_i\}|$, i.e., the number of wedges α_i which contain α. It

follows that the previously introduced γ_i and γ_j are precisely wedges whose multiplicity is 1 and which are antipodal (i.e., they are crossed by the same straight line). It is not difficult to see that the arguments can be reversed, thus proving the following theorem:

Theorem. A simple polygon P is monotone if and only if the polar diagram of its arcs contains at least one pair of antipodal intervals of multiplicity 1.

This theorem immediately suggests an algorithm to test a simple polygon P for monotonicity: we process the boundary of P in the order $e_0, e_1, \ldots, e_{n-1}$. When we process an edge e_j, we insert α_i into the polar diagram by updating the multiplicities of the polar wedges so far constructed. Note that the multiplicity of a wedge cannot decrease; since we are seeking polar wedges of multiplicity 1, it is irrelevant whether a wedge has multiplicity 2 or greater. Thus we shall label each wedge with a symbol in the set $\{0,1,2\}$, where $\{0,1\}$ are actual multiplicities and 2 denotes a multiplicity ≥ 2.

During processing we maintain a doubly-linked circular list of polar angles, each of which separates two adjacent wedges. Each of the two pointers (forward and backward) is labeled in the set $\{0,1,2\}$. In addition, we have a pointer to the current position θ in the polar diagram. We claim that the list satisfies the following properties:

(1) the angles are in increasing counterclockwise order;

(2) the wedge labels — possibly with the exception of one single 0 label — form an alternating string of 1's and 2's.

To prove this claim, we outline the algorithm.

Initial step. 0 is chosen conventionally as θ_0. There is a single wedge, labeled 0. We insert into the list angle θ_1 and label with 1

the wedge determined by α_1, and θ is set to θ_1.

<u>General step</u>. Let θ be the current position and assume that the list satisfies properties (1) and (2). We process α_i. If θ_i is larger than θ we scan the list forward, while if θ_i is not larger than θ we scan it backwards. The scan terminates when θ_i can be inserted. In this process we increase by 1 each wedge label different from 2 and merge any two consecutive wedges receiving identical labels (merging is, of course, done by deleting the node corresponding to the angle value which separates them). With regard to the updating of θ, suppose that θ_i is to be inserted into wedge $[\beta,\beta']$: if the pointer from β to β' is labeled 0 or 1, then a new list node is created and $\theta \leftarrow \theta_i$; else no new node is created and $\theta \leftarrow \beta'$.

Clearly property (1) is satisfied after the general step, because θ_i is inserted in its appropriate order. Property (2) is also satisfied, since wedge merging guarantees the alternation of 1 and 2 labels on continguous wedges (with labels different from 0).

From the performance viewpoint, it is convenient to charge the computational work to each individual list node. A list node is initially established in constant time. Subsequently, during list scans, a node is traversed in one direction; its pointers are for brevity referred to as <u>incoming</u> and <u>outgoing</u>. The labels of both pointers are updated $(0 \rightarrow 1, 1 \rightarrow 2, 2 \rightarrow 2)$ and when both pointers are labeled 2 the node is deleted. Each node traversal uses constant time and each node can be traversed at most twice before its deletion. It follows that the total running time is $O(n)$.

At the termination of the above algorithm, we have a partition of the polar range $[0,2\pi)$ into $O(n)$ wedges with alternating labels 1 and 2.

Scanning the sequence of angles by means of two pointers b_1 and b_2 we can determine the pairs of antipodal wedges.[1] Specifically, let $\theta(b_1)$ denote the angle pointed to by b_i. We set initially $\theta(b_i) = 0$ and advance $\theta(b_2)$, until $\theta(b_2) - \theta(b_1) \geq 180°$; at this point $\theta(b_1)$ is advanced until $\theta(b_2) - \theta(b_1) < 180°$, when the advancement of $\theta(b_2)$ is resumed; and so on until $\theta(b_2) = 0$. This process clearly runs in time $O(n)$ and obtains all pairs of antipodal wedges (which are known to be $O(n)$ [3]), whose labels are concurrently compared. Since both major tasks (construction of the sectors and detection of antipodal pairs) can be completed in linear time, the entire monotonicity test runs in linear time, which is optimal.

Note that the above algorithm obtains <u>all</u> directions with respect to which P is monotone.

Conclusion

Testing an arbitrary polygon (i.e., a sequence of vertices) for convexity [3], testing a simple polygon for star-shapedness [2], and testing a simple polygon for monotonicity are all $\Theta(n)$ time problems. An interesting open problem in this area is testing an arbitrary polygon for simplicity. For this problem, the fastest algorithm known is $\Theta(n\log n)$ time [4], but no super-linear lower bound is known.

Acknowledgements

The authors thank G. T. Toussaint and H. El-Gindy for helpful discussions on this problem.

---

(1) The following technique is a modification of an algorithm due to M. I. Shamos [3] to obtain the diameter of a convex polygon.

REFERENCES

[1] Garey, M. R., D. S. Johnson, F. P. Preparata, and R. E. Tarjan, "Triangulating a simple polygon," Info. Proc. Letters, Vol. 7, No. 4, June 1978, pp. 175-179.

[2] Lee, D. T., and F. P. Preparata, "An optimal algorithm for finding the kernel of a polygon," JACM, Vol. 26, 1979, pp. 415-421.

[3] Shamos, M. I., "Computational geometry," Ph.D. thesis, Dept. of Computer Science, Yale University, 1978.

[4] Shamos, M. I. and D. Hoey, "Geometric intersection problems," 17th Annual Symposium on Foundations of Computer Science, pp. 208-215, October 1976.

An improved algorithm for the rectangle enclosure problem[+]

D. T. Lee and F. P. Preparata

January, 1981

Abstract. Given a set of n rectangles in the plane, with sides parallel to the coordinate axes, the rectangle enclosure problem consists of finding all q pairs of rectangles such that one rectangle of the pair encloses the other. In this note we present an algorithm alternative to the one by Vaishnavi and Wood; while both techniques have worst-case running time $O(n\log^2 n + q)$, ours uses optimal storage $O(n)$ rather than $O(n\log^2 n)$ as the Vaishnavi-Wood's technique. Our algorithm works entirely in-place and uses very conventional data structures.

[+]This work was partially supported by the National Science Foundation under Grants MCS 78-13642 and MCS 79-16847 and by the Joint Services Electronics Program under Contract N00014-79-C-0424.

AN IMPROVED ALGORITHM FOR THE RECTANGLE ENCLOSURE PROBLEM

1. Given a set of n rectangles in the plane, with sides parallel to the coordinate axes (iso-oriented rectangles), the rectangle enclosure problem consists of finding all q pairs of rectangles such that one rectangle of the pair encloses the other.

This problem is an interesting one in the "geometry of rectangles", which is relevant to several practical applications, primarily to the computer-aided design of VLSI circuits [1,2]. The best known solution is due to Vaishnavi and Wood [3] and runs in time $O(n\log^2 n + q)$ using space $O(n\log^2 n)$; their approach makes crucial use of some versatile, but space-consuming, data structures called range trees and segment trees [4,5].

In this paper a new approach is described which achieves the same time bound but uses only linear space. Thus it is space-optimal; as to computation time, there is still a gap between upper- and lower-bound.

2. We begin by transforming the rectangle enclosure problem into an equivalent one, which is easier to describe and understand. Let $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ be a set of iso-oriented rectangles in the plane (x,y), where $r_i = [x_1^{(i)}, x_2^{(i)}] \times [y_1^{(i)}, y_2^{(i)}]$, with $x_1^{(i)} < x_2^{(i)}$ and $y_1^{(i)} < y_2^{(i)}$. Rectangle r_i encloses rectangle r_j if the following four conditions hold:

$$(1) \qquad x_1^{(i)} \le x_1^{(j)},\ x_2^{(j)} \le x_1^{(i)},\ y_1^{(i)} \le y_1^{(j)},\ y_2^{(j)} \le y_2^{(i)}$$

These conditions are trivially equivalent to

$$(2) \qquad -x_1^{(j)} \le -x_1^{(i)},\ x_2^{(j)} \le x_1^{(i)},\ -y_1^{(j)} \le -y_1^{(i)},\ y_2^{(j)} \le y_2^{(i)}$$

which express the well-known relation "$\prec$" of <u>dominance</u> between two four-dimensional points, that is, $(-x_1^{(j)}, x_2^{(j)}, -y_1^{(j)}, y_2^{(j)}) \prec (-x_1^{(i)}, x_2^{(i)}, -y_1^{(i)}, y_2^{(i)})$. Thus, after mapping each $r_i \in \mathcal{R}$ into its corresponding four-dimensional point, the rectangle enclosure problem becomes the <u>point dominance problem</u> in 4-space. Specifically: "Given a set $S = \{p_1,\ldots,p_n |\ p$ is a point in 4 space$\}$ for each point $p_i \in S$ find a set $S_i \subseteq S$ such that $S_i = \{p | p \in S, p \prec p_i\}$".

Our approach to solving the point dominance problem is very similar to the one used in [6] to solve a closely related problem, finding the <u>maxima</u> of a set of vectors (i.e., the subset $M \subseteq S$ defined as $M = \{p | p \in S$ and there is no $q \in S$ such that $p \prec q\}$). The technique is an application of the divide-and-conquer principle. Let u_1, u_2, u_3, u_4 be the coordinates of our 4-space. The elements of S are reindexed so that $(i < j) \Rightarrow (u_1(p_i) \le u_1(p_j))$. We then have:

<u>Algorithm Dominance</u>

D1. (Divide) Partition S into S_1 and S_2, where $S_1 = \{p_1,\ldots,p_{\lfloor n/2 \rfloor}\}$ and $S_2 = \{p_{\lceil n/2 \rceil},\ldots,p_n\}$.

D2. (Recur) Solve the point-dominance problem on S_1 and S_2, separately.

D3. (Merge) Find all the pairs $p_i \prec p_j$, where $p_i \in S_1$ and $p_j \in S_2$.

We shall now discuss the implementation of step D3. For $p_i \in S_1$ and $p_j \in S_2$, since $u_1(p_i) \le u_1(p_j)$ by construction, we have $p_i \prec p_j$ if and only if $u_\ell(p_i) \le u_\ell(p_j)$ for $\ell = 2,3,4$. Thus Step D3 is in effect, a three-dimensional problem. Here again, we solve it by a divide-and-conquer technique. Specifically, let $\bar{u}_2$ be the median of $\{u_2(p_i) | p_i \in S_2\}$.

<u>Algorithm Merge</u>

M1. (<u>Divide</u>) Partition S_1 into $\{S_{11}, S_{12}\}$ and S_2 into $\{S_{21}, S_{22}\}$, so that $S_{11} = \{p \mid p \in S_1, u_2(p) \le \bar{u}_2\}$, $S_{21} = \{p \mid p \in S_2, u_2(p) \le \bar{u}_2\}$, and $S_{12} = S_1 - S_{11}$, $S_{22} = S_2 - S_{21}$.

M2. (<u>Recur</u>) Solve the merge problem on the set pairs $\{S_{11}, S_{21}\}$ and $\{S_{12}, S_{22}\}$.

M3. (<u>Combine</u>) Find all pairs $p_i \prec p_j$ such that $p_i \in S_{11}$ and $p_j \in S_{22}$.

To convince ourselves of the correctness of the approach, note that S has been partitioned into $\{S_{11}, S_{12}, S_{21}, S_{22}\}$. Within each of these four subset, the point-dominance problem is solved in D2; it remains to be solved between pairs of subsets. Of the six pairs, $\{S_{11}, S_{12}\}$ and $\{S_{21}, S_{22}\}$ are also processed in D2; $\{S_{11}, S_{21}\}$ and $\{S_{12}, S_{22}\}$ are processed in M2; $\{S_{11}, S_{22}\}$ are processed in M3, while $\{S_{12}, S_{21}\}$ need not be considered because for each $p_i \in S_{12}$ and $p_j \in S_{21}$ we have $u_1(p_i) \le u_1(p_j)$ and $u_2(p_i) > u_2(p_j)$. Notice, also that Step M3 (Combine) is a two-dimensional Merge problem (in u_3 and u_4).

The key operation of the entire task is therefore the implementation of step M3, the two-dimensional Merge (Combine). Indeed the entire computation reduces to the careful sequencing of steps like M3; therefore, in what follows we shall concentrate on devising an efficient implementation of "Combine". We shall show that "Combine" can be done in time linear in the input size, after an initial O(nlogn) sorting, which is charged to the entire point-dominance problem.

3. The initial preprocessing consists in preparing a suitable data structure for the set S. Specifically, we set up a <u>quadruply-threaded list</u> (QTL), with bidirectional links. For each $p \in S$, we construct a node containing the information $(u_1(p), u_2(p), u_3(p), u_4(p))$; after sorting S on

each coordinate, we establish four pointers NEXT1,...,NEXT4, so that NEXTj describe the ordering on u_j. Bidirectional links are established by four additional pointers PREDj (j = 1,2,3,4). The setting up of the QTL for S, obviously, uses time O(nlogn).

The QTL lends itself, very naturally, to the linear-time implementation of the set-splitting operations specified by steps D1 and M1 of the preceding algorithms. Indeed, suppose we want to split S into $\{S_1,S_2\}$ and that the elements of, say, S_1, are marked. Then, by traversing the QTL on a selected pointer NEXTi, the list corresponding to this pointer is easily "unmerged" into two lists, corresponding to the two sets $\{S_1,S_2\}$ of the partition. Analogously given S_1 and S_2, in linear time we can merge the two corresponding lists using "natural merge" [7]. Note that splitting and merging operations simply involve modification of the pointers and use no additional space for storing data.

Let us now consider the implementation of Step M3, "Combine". Here we have two sets, S_{11} and S_{22}, of two-dimensional points. The sets are actually represented as a doubly-threaded list (that is, threaded on the two coordinates u_3 and u_4); BEG31 and BEG32 denote pointers to the first positions of the two lists, for S_{11} and S_{22}, respectively, corresponding to coordinate u_3 (which is the coordinate to be scanned). We also make use of a new list L, which is destined to contain the sorted sequence of the u_4-coordinates of a subset of S_{11} (specifically, the u_4-coordinates of the points of S_{11} whose u_3-coordinate is no larger than the current scan value). Temporarily, we use NEXTL and BEGL to denote the forward and initial pointers for L, although — as we shall see below — NEXT4 can be used in place of NEXTL. Letting $|S_{22}| = s$ we propose the following algorithm:

Algorithm Combine

```
1    begin j1 ←BEG31, j2 ← BEG32
2          while (j2 ≤ s) do
3                begin if (u3[j1] ≤ u3[j2]) then
4                         begin insert u4[j1] into L
5                               j1 ← j1+1
                          end
                     else begin ℓ ← BEGL
7                               while (ℓ≠Λ) and (u4[j2] > u4(ℓ)) do
8                                     begin print (j2,ℓ)
9                                           ℓ ← NEXTL[ℓ]
                                      end
10                               j2 ← j2+1
                          end
                 end
end
```

The above algorithm has obviously the structure of a merge technique. In step 3 we test whether we should advance on S_{11} or on S_{22}. In the former case we must insert $u_4[j_1]$ into L (Step 4). In the latter case (Steps 6-9), we scan the list L from its smallest element, thereby determining all the points dominated by p_{j_2}; this part of the procedure is straightforward and runs in time proportional to the number of pairs (j_2,ℓ) which are printed. The crucial task of the procedure is represented by Step 4: "insert $u_4[j_1]$ into L". Indeed, at first sight, it appears to globally require time proportional to $|S_{11}|^2$, since each insertion may require a full scan of L; a more sophisticated implementation of L with an AVL tree would cut the global time down to $(|S_{11}|\log|S_{11}|)$. However, there is an interesting way to organize Step 4 so that its global time requirements be $O(|S_{11}|)$. This is accomplished by a backward pre-scan of the u_3-list of S_{11}, which generates the schedule of insertion into L of the terms of the u_4-list of S_{11}. Indeed, starting from the largest element of the u_3-list and proceeding towards the smallest, let $u_3(j)$ be the element currently considered in the scan: we save the <u>current</u> value of PRED4[j] (on the u_4-list) and update the u_4-list by deleting $u_4[j]$. It is clear

that PRED4[j] thus saved will give — in constant time — the place of insertion of $u_4[j]$ when the u_3-list of S_{11} is scanned forward. In summary, the insertion schedule is obtained by the following algorithm:

```
    begin ℓ ← LAST (u3 list)
          while (PRED3[ℓ] ≠ BEG) do
    begin NEXT4[PRED4[ℓ]] ← NEXT4[ℓ]
          PRED4[NEXT4[ℓ]] ← PRED4[ℓ]
          ℓ ← ℓ - 1
    end
end
```

Example: Given the set S_{11} depicted in figure 1(a), in figure 1(b)

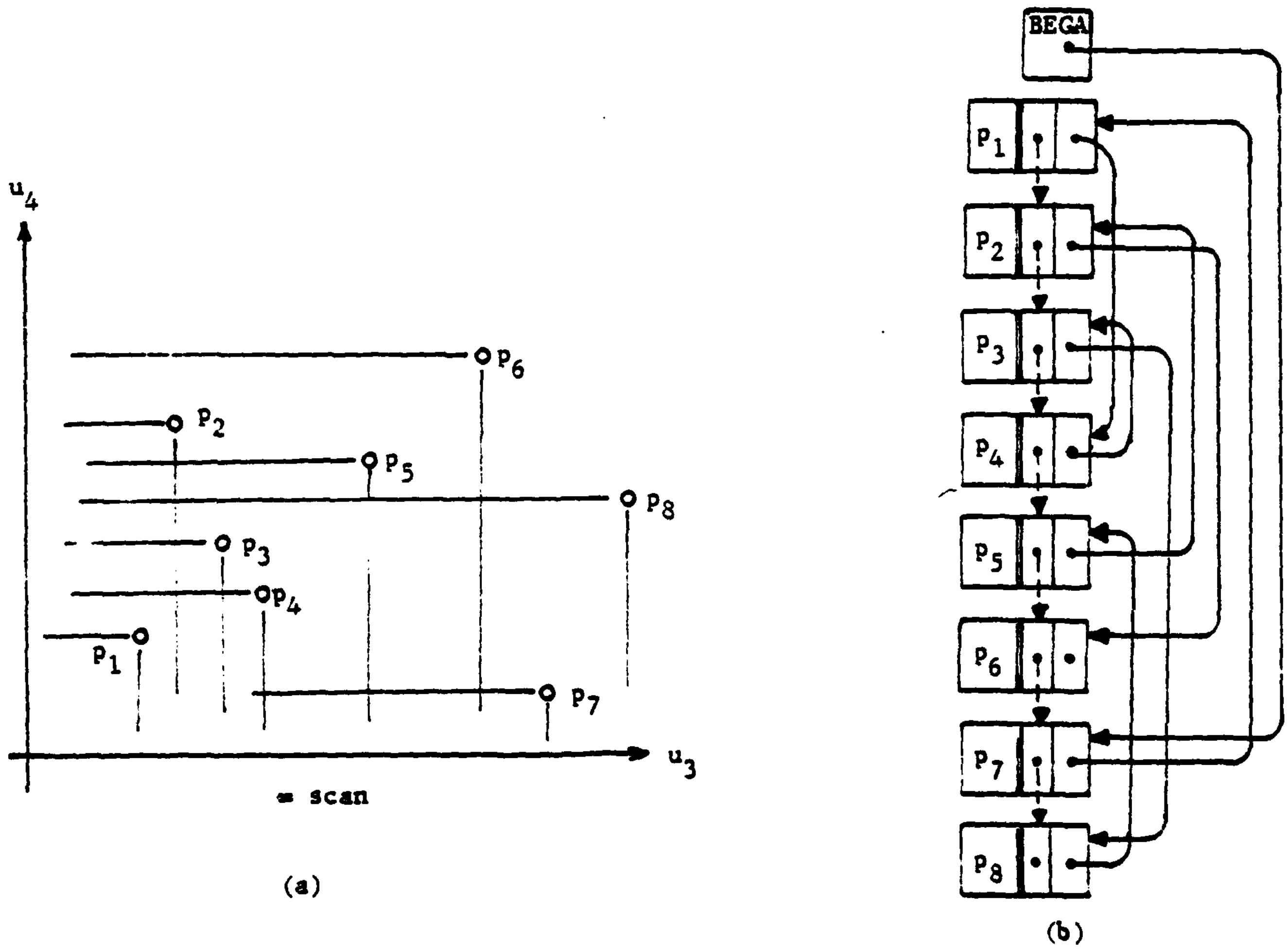

Figure 1. An example of set $S_{11} = \{p_1, \ldots, p_8\}$ and of the associated doubly-threaded list. NEXT3 links are shown by broken lines; NEXT4 links by solid lines.

we illustrate the initial configuration of the u_3- and u_4-list. The initial configuration of the array PRED4 is:

j	1	2	3	4	5	6	7	8
PRED4:	7	5	4	1	8	2	BEG	3

The evolution of this array when executing the above scan is shown compactly below (entries being updated are encircled)

j	1	2	3	4	5	6	7	8	After Scanning
initial PRED4	7	5	4	1	8	2	BEG	3	—
	7	5	4	1	3	2	BEG	3	p_8
	BEG	5	4	1	3	2	BEG	3	p_7
	BEG	5	4	1	3	2	BEG	3	p_6
	BEG	3	4	1	3	2	BEG	3	p_5
	BEG	3	1	1	3	2	BEG	3	p_4
	BEG	1	1	1	3	2	BEG	3	p_3
final (insertion schedule)	BEG	1	1	1	3	2	BEG	3	p_2

Therefore, the final configuration of the array PRED4 completely specifies the insertion schedule into the L-list (which becomes the u_4-list when the scan is complete) and line 4 of COMBINE can be executed in constant time. This shows that the entire COMBINE procedure runs in time linear in $|S_{11}| + |S_{22}|$ and in the number of pairs (point dominances) obtained.

4. To analyze the performance of the proposed technique we note:

1) All processing occurs in place, uses the QTL arrays, and reduces to transformations of the pointers' values. Thus the space used is O(n).

2) As regards processing time each dominance pair (i.e., each enclosed pair of rectangles) is found exactly once and in constant

time by the while-loop (7-9) of Combine. Thus, if q is the number of pairs, O(q) optimal time is used for this activity. The remaining computing time depends exclusively on the size n of S: denote it by D(n). Also denote by $M_d(r,s)$ the running time of Algorithm Merge on two sets with r and s d-dimensional points, respectively ($d = 2,3$). Assuming, for simplicity, that n be even, we have

$$(3) \qquad D(n) = 2D(n/2) + M_3(n/2, n/2) + O(n)$$

where O(n) is the time used by the "divide" step D1. Analogously, we have (assume that $|S_{21}| = m$ and that r be even):

$$(4) \qquad M_3(r,s) = M_3(r/2,m) + M_3(r/2,s-m) + M_2(r/2,\max(m,s-m)) + O(r+s)$$

where, again, $O(r+s)$ time is needed to perform the set split. An upper-bound to $M_3(r,s)$ is obtained by maximizing the right-side of (4) with respect to m. Since $M_2(r',s')$ is $O(r'+s')$, arguing as in [6], we obtain that $M_3(r,s) = O((r+s)\log(r+s))$ and, consequently, that $D(n) = O(n(\log n)^2)$.

Incidentally, the 3-dimensional dominance problem is implicitly solved by the technique described in this paper. In other words, given a set of n points in 3-space, the p dominance pairs existing in this set can be found in time $O(n\log n + p)$ and space $O(n)$, both of which are optimal (see [6]).

References

[1] H. S. Baird, "Fast algorithms for LSI artwork analysis," Design Automation and Fault-Tolerant Computing, 2, pp. 179-209; (1978).

[2] U. Lauther, "4-dimensional binary search trees as a means to speed up associative searches in the design verification of integrated circuits," Jour. of Design Automation and Fault-Tolerant Computing, Vol. 2, n. 3, pp. 241-247; July 1978.

[3] V. Vaishnavi and D. Wood, "Data structures for the rectangle containment and enclosure problems," Computer Graphics and Image Processing, 13, pp. 372-384; (1980).

[4] J. L. Bentley and T. Ottmann, "Algorithms for reporting and counting geometric intersections," IEEE Transactions on Computers, vol. 28, n. 9, pp. 643-647; September 1979.

[5] H.-W. Six and D. Wood, "The rectangle intersection problem revisited," Comp. Sci. Tech. Report 79-CS-24, McMaster University; 1979.

[6] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," Journal of the ACM, vol. 22, no. 4, pp. 469-476, October 1975.

[7] D. E. Knuth, The Art of Computer Programming, vol. 1, Sorting and Searching, Addison-Wesley, Reading, Mass. 1972.

SHORTEST PATHS WITHIN A SIMPLE POLYGON

F. P. Preparata

This note describes an efficient solution of the following geometric problem: given a simple n-vertex polygon P in the Euclidean plane and two distinguished points s and t, respectively called <u>source</u> and <u>destination</u>, in the interior of P, find the shortest path between s and t lying entirely within P.

This problem has been previously considered by M. I. Shamos [1], who called it "internal distance" and described an algorithm which solves it in time $O(n^2)$. Shamos' method is based on the prior construction of the so-called <u>viewability graph</u> of a polygon, namely the set of edges which join pairs of vertices of the polygon and are entirely contained in its interior; once the viewability graph is obtained, the shortest path within the polygon is the shortest path on the viewability graph when each edge is weighted with its length. We shall now show that only relevant portions of the viewability graph need be constructed thereby reducing the computation time from $O(n^2)$ to $O(n\log n)$.(1)

We need some nomenclature.

<u>Definition 1</u>. An n-vertex simple polygon $P = (q_1, q_2, \ldots, q_n)$ is a closed polygonal chain such that no two nonconsecutive edges intersect. A <u>diagonal</u> of P is a line segment $\overline{q_i q_j}$, $j \neq i+1$, which does not cross any edge of P. P is said to be triangulated if its interior has been divided

This work was supported in part by National Science Foundation under Grant MCS 7 -13642 and in part by the Joint Services Electronics Program under Cont. .ct DAAG-29-78-C-0016.

(1) All logarithms are to the base 2.

into n-2 triangles by n-3 diagonals.

Definition 2. The dual tree of a triangulated simple polygon P is a graph T = (V,E) such that each vertex of V corresponds to a triangle of the triangulation and each edge of E connects two vertices of V if and only if the corresponding two triangles share a diagonal of P. The diagonal of P and the corresponding edge in T are said to be dual.

Obviously T is a tree whose vertices have degree at most 3.

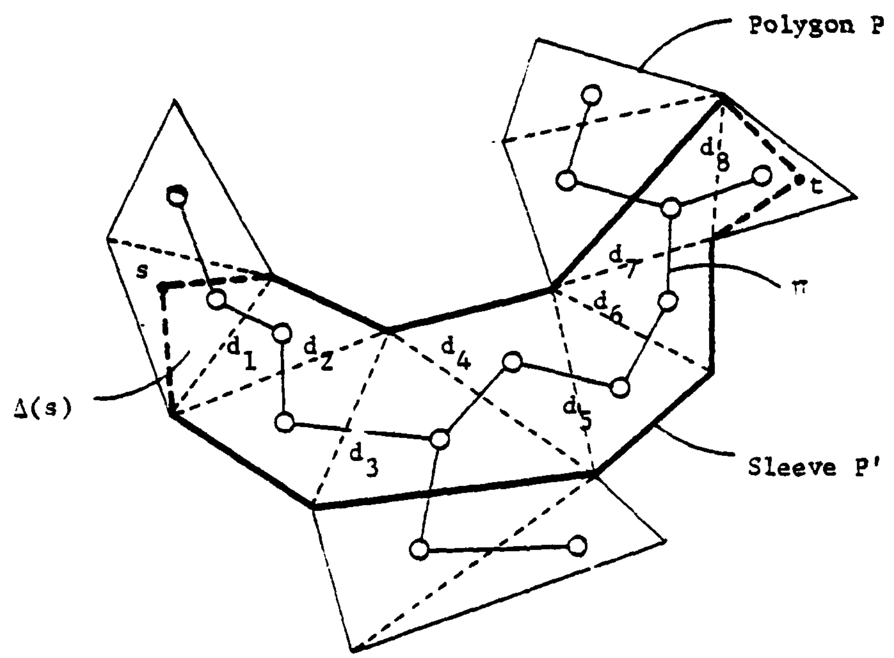

Figure 1. Illustration of polygon, sleeve, diagonals, and dual path π.

Definition 3. A triangulated polygon is called a sleeve if its dual graph is a polygonal chain. Figure 1 illustrates the notions of triangulated polygons, diagonals, sleeves, and dual graphs.

Our method is based on the following observation. Let $\Delta(s)$ and $\Delta(t)$ be the two triangles in (the triangulated) P which contain s and t, respectively. In T there is a unique path π between the vertices which are the duals of $\Delta(s)$ and $\Delta(t)$. The edges in π are themselves duals of diagonals of P, so that the sequence of edges of π corresponds to a sequence of diagonals $d_1, d_2, \ldots, d_p$ (ordered from s to t). Since d_i divides P into two parts, which respectively contain s and t, the shortest path from s to t within P crosses each and every $d_1, \ldots, d_p$. Notice that any other diagonal of P is either wholly contained in the shortest path or does not share any internal point with it, since the shortest path is entirely contained in the triangles which are duals of the vertices of π.

This also indicates that, without loss of generality, we may restrict ourselves to the plane polygon P' which dualizes to π, with the further condition that s and t be themselves vertices of the polygon (that is, we replace $\Delta(s)$ with the triangle having as its vertices s and the extremes of d_1; similarly $\Delta(t)$ is replaced by the triangle having as its vertices t and the extremes of d_p). The plane polygon P' in fact is a sleeve by Definition 3. Hereafter we assume that the given polygon P is a sleeve with n vertices, including s and t.

Let $v_i^{(1)}$ and $v_i^{(2)}$ be the two extreme points of diagonal d_i, $1 \leq i \leq n-3$, and let $D(s,v_i^{(j)})$ be the shortest path from s to $v_i^{(j)}$, $j = 1,2$, within the polygon P. It is easy to show that $D(x,v_i^{(j)})$ is a polygonal chain whose points are vertices of P. Let $D_i \triangleq D(s,v_i^{(1)}) \cup D(s,v_i^{(2)})$. In the graph D_i there is a unique vertex v which is common to both $D(s,v_i^{(1)})$

and $D(s,v_i^{(2)})$ and is farthest from s on either chain; we say that the two chains diverge at v and obviously $D(v,v_i^{(1)})$ and $D(v,v_i^{(2)})$ have no edge in common.

Assume at first that neither of the latter subchains is empty; then we claim that $D(v,v_i^{(j)})$ $(j = 1,2)$ is an inward-convex polygonal chain, i.e., it is convex with convexity facing toward the interior of P. To prove this, we first show that the region R_i delimited by $D(v,v_i^{(1)})$, $D(v,v_i^{(2)})$, and d_i (briefly called a funnel) is entirely contained in P. Let $d_s,d_{s+1},\ldots,d_{i-1}$ be the diagonals crossed by $D(v,v_i^{(1)})$ and $D(v,v_i^{(2)})$. Clearly the triangle $(v,v_s^{(1)},v_s^{(2)}) = R_s$ is contained in P; assuming inductively that $R_{i-1} \subset P$, we see that R_i is obtained by adjoining to R_{i-1} all or part of a triangle contained in P, thus showing that $R_i \subset P$. Next if $D(v,v_i^{(j)})$ is not inward-convex, then, by the triangle inequality, there is a shorter path from v to $v_i^{(j)}$, entirely contained P, thereby violating the hypothesis that $D(s,v_i^{(j)})$ is a shortest path from s to $v_i^{(j)}$ (see Figure 2). This convexity property also proves that $D(s,v_i^{(1)})$

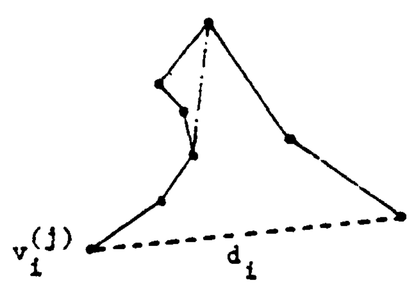

Figure 2. Illustration of inward-convexity of $D(s,v_i^{(j)})$.

and $D(s,v_i^{(2)})$ may diverge at most at one vertex v; for, if they diverge at some other vertex u_1, then they must reconverge at some vertex u_2, and the two distinct subchains from u_1 and u_2 must both be inward-convex, a clear inconsistency.

In general D_i is a (possibly empty) chain branching at some vertex v, called a cusp into two inward-convex chains, which delimit a (possibly degenerate) funnel. Notice that either of these two chains could be empty (but not both, since $v_i^{(1)} \neq v_i^{(2)}$). If, say, $D(v,v_i^{(1)})$ is empty, then clearly $D(v,v_i^{(2)}) = d_i$; in this case the funnel degenerates to a single diagonal, R_i has no interior, and D_i becomes a single chain.

The algorithm successively constructs $D_1, D_2, \ldots, D_p$ and finally $D(s,t)$. In detail we have:

Initial Step. Construct D_1 by connecting s to $v_1^{(1)}$ and $v_1^{(2)}$.

General Step. (Construct D_{i+1} from D_i). Let v be the cusp of D_i, at which the two subchains $\overline{u_a u_{a+1} \ldots u_b}$ and $\overline{u_a u_{a-1} \ldots u_0}$ diverge, where $v = u_a$ $v_i^{(1)} = u_b$, $v_i^{(2)} = u_0$. Without loss of generality, let $v_i^{(1)} = v_{i+1}^{(1)}$ (see Figure 3). Starting from u_0 scan the sequence $u_0, u_1, \ldots, u_b$ and let j be the smallest integer for which $\overline{v_{i+1}^{(2)} u_j}$ becomes a supporting[(1)] segment of the boundary of R_i. We distinguish two cases

(1) $j \leq a$. Delete all edges $\overline{u_\ell u_{\ell+1}}$ for $0 \leq \ell \leq j-1$ and add edge $\overline{u_j v_{i+1}^{(2)}}$.

(2) $j > a$. Delete all edges $\overline{u_\ell u_{\ell+1}}$ for $0 \leq \ell \leq j-1$ and add edge $\overline{u_j v_{i+1}^{(2)}}$; u_j becomes the cusp of R_{i+1}.

[(1)] A line ℓ is a supporting line of a convex open curve C if it has at least one point in common with C and C lies all on one side of ℓ, with its convexity facing ℓ.

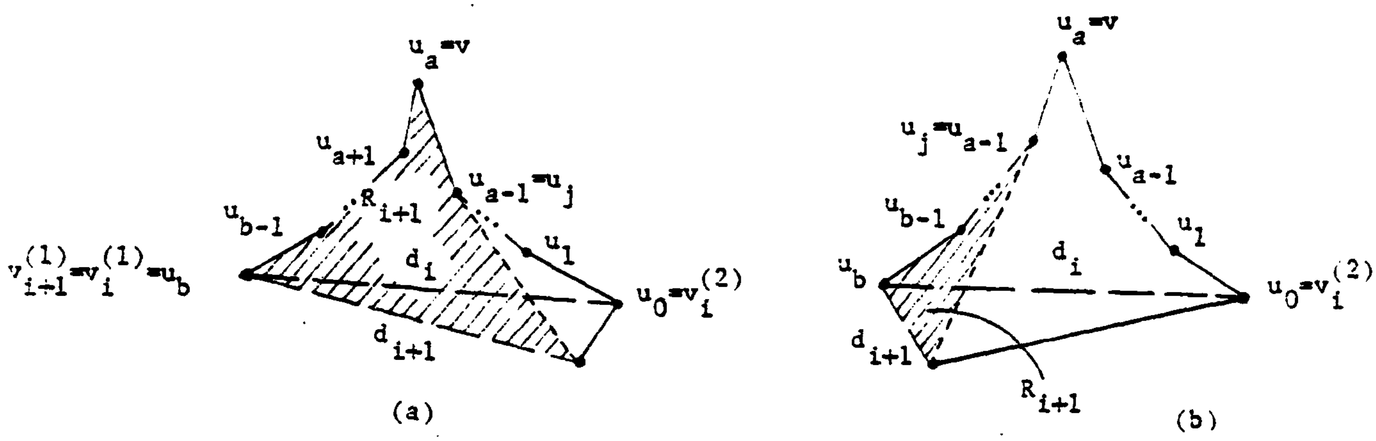

Figure 3. Illustration of the general step. In (a), u_j belongs to $\overline{u_s \ldots u_a}$; in (b) u_j belongs to $\overline{u_a \ldots u_b}$. R_{i+1} is shown cross-hatched.

<u>Final Step</u>. Once D_{n-3} has been constructed, one of the two sides of P incident on t is treated as a diagonal d_{n-2} and the general step is applied to this case, yielding D(s,t).

The correctness of the algorithm depends upon the following fact. For any point u in the triangle defined by the two diagonals d_i and d_{i+1}, a shortest path from s to u passes through v. For, assume the contrary. If both $D(v,v_i^{(1)})$ and $D(v,v_i^{(2)})$ are nonempty, consider the edge incident on v on either of these subchains: since P is a sleeve, one of them is a diagonal of P (although not necessarily an original diagonal of the triangulated P); if either of these subchains is empty, then, as we saw earlier, the other subchain consists of a single diagonal. In either case, let $\overline{v\ v'}$ be this diagonal and let $\overline{v\ v''}$ be the other edge (Figure 4).

The polygonal chain $\ell(s,u)$ which defines a shortest path from s to u crosses $\overline{v\,v'}$ at some point $p \neq v$. We claim that the distance ℓ_1 from s to p on $\ell(s,u)$ is strictly less than that (called ℓ_2) on the polygonal chain obtained by concatenating $D(s,v)$ and the segment $\overline{v\,p}$. To prove this, note that the wedge formed by $\overline{v\,v'}$ and $\overline{v\,v''}$ intersects both d_i and d_{i+1}; thus, the destination point u in the triangle defined by d_i and d_{i+1} is in one of three regions (see Figure 4); all the three cases, however, are treated analogously. Assuming, for example, that $\ell(s,u)$ crosses $\overline{v\,v''}$ (case shown in Figure 5) in a point p_1, we have, by hypothesis, that $\ell(s,u)$ is a shortest path from s to u

$$\ell_1 + \text{length}(\ell(p,p_1)) < \ell_2 - \text{length}(\overline{vp}) + \text{length}(\overline{vp_1})$$

where $\ell(p,p_1)$ is the subchain of $\ell(s,u)$ from p to p_1. But, by the triangle inequality, $\text{length}(\overline{vp_1}) \leq \text{length}(\overline{vp}) + \text{length}(\ell(p,p_1))$, whence

$$\ell_2 - \ell_1 > \text{length}(\overline{vp}) + \text{length}(\ell(p,p_1)) - \text{length}(\overline{vp_1}) \geq 0$$

i.e., $\ell_2 > \ell_1$. Therefore $\ell_2 + \text{length}(\overline{pv'}) > \ell_1 + \text{length}(\overline{pv'})$, contradicting the known fact that the shortest path from s to v' passes through v.

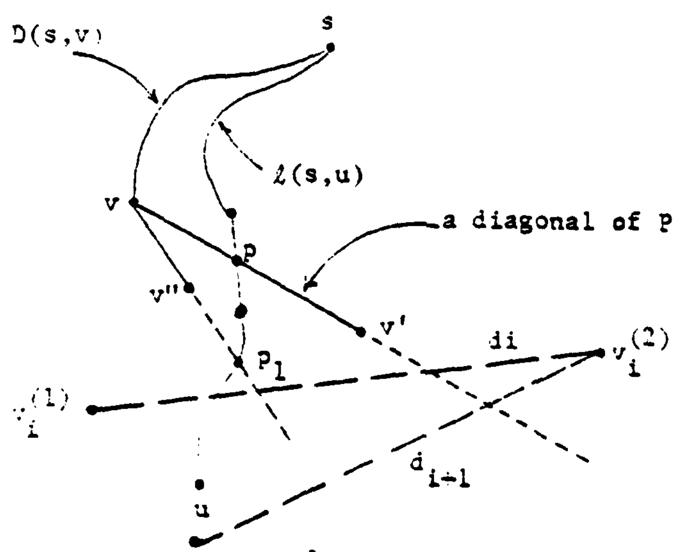

Figure 4. Illustration for the proof that a shortest path between s and u passes through v.

We now analyze the running time of the algorithm. Case (1) of the general step takes constant time; Case (2) may involve scanning a large number of vertices; however, once a vertex has been scanned and the corresponding angle has been found to require continuation of the scanning process, that vertex is definitively eliminated from consideration. Since in P there are n-2 vertices besides s and t, the entire algorithm runs in time O(n). The shortest-path algorithm, however, assumes that P be a sleeve. To transform an arbitrary simple n-vertex polygon into a sleeve, we first triangulate it in time O(nlogn) using the algorithm of [2]; the dual T of the given polygon is obtained in time O(n) and, still in linear time, the path π is obtained. This completes the transformation of the polygon into the required sleeve. Thus the entire procedure runs in time O(nlogn), the triangulation task being dominant. However, if preprocessing is allowed, the shortest path problem can be solved in O(n) time for every pair of points s and t. We summarize the results as a theorem below.

Theorem 1. Given a simple polygon P with n vertices and two points s and t in the interior of P, a shortest path between s and t lying entirely within P can be found in O(nlogn) time. If preprocessing of the polygon P is allowed with preprocessing time O(nlogn), then the problem can be solved in O(n) time for any two points s and t in the interior of P.

References

1. M. I. Shamos, Computational Geometry, Dept. of Comp. Sci., Yale University, 1977. To be published by Springer Verlag.

2. M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, "Triangulating a simple polygon," Information Processing Letters, Vol. 7, No. 4, pp. 175-179, June 1978.

LMED

8